Journal of Membrane Computing (2019) 1:52–57
https://doi.org/10.1007/s41965-019-00007-0

REGULAR PAPER

Inference of bounded L systems with polymorphic P systems

Gábor Román[1]

Received: 20 August 2018 / Accepted: 4 January 2019 / Published online: 22 January 2019

Abstract
In this paper, we are going to solve the inference problem of bounded L systems, namely such L systems which work on filaments having length up to a fixed size. We will show that these bounded L systems have considerable computational power as they can simulate linear-bounded automata. To carry out the inference, we are going to construct a specific polymorphic P system with target indication, which can reproduce the transitions of the examined bounded L system, and which is of size $O(n|G|^4)$, where G is the alphabet of the bounded L system with n as the maximal size of the filaments.

Keywords L-systems · Polymorphic P systems · Linear-bounded automata · Inference

1 Introduction

The inference problem was formulated for learning languages, grammar from samples and for different families of automata from their computations. Given an automaton, one could apply its rules starting from initial configurations to gain computations of the automaton. During inference one solves the inverse process, namely gaining the rules and the structure of an unknown automaton from its given computations. For a comprehensive view of the inference problem, see, for example, [6, 8], or the proceedings of the International Colloquium on Grammatical Inference, see, for example, [21].

In this paper, we are going to consider the inference problem of L systems. A thorough investigation of this topic can be found in [5], and there is also a survey which covers this area up to 2009, see [3]. L systems were introduced by Lindenmayer in [12] to study cellular development with mathematical models. These systems had a great impact since, see [17–20]. They in fact have quite great computing ability as it is described in [7], where the author showed that for every Turing machine there exists a Lindenmayer system which simulates it.

Membrane computation and P systems were originally introduced by Păun in 1998, see [14]. For a comprehensive introduction to this field, see the books [15, 16]. The most up-to-date results can be found on the P system web site [22], and at the bulletin of the International Membrane Computing Society [4]. Polymorphic P systems are special variants of P systems, which deduce the applied rules from the contents of their membranes, which ability combined with communication through target indication makes them good candidates for solving inference-related problems. These systems were first described in 2011, see [1]. For a survey about the development of these systems up to 2016, see [2].

Our goal is to give a method to construct polymorphic P systems which solve the inference problem of L systems. One problem arises, namely that L systems work on strings while membrane systems, although may process strings, mainly work on objects in their compartments. We concentrate on the case when the contents of the membranes are multisets. The missing information here is the sequentiality, which can be represented in the membrane structure or object encoding. Instead of representing sequentiality through some intricate method using the membrane structure, we look at the object encoding method.

The investigated representation is the encoding through indexing the letters in the filaments. Let G be the nonempty finite alphabet of the L system and define $\Sigma_\alpha := \{a_i | a \in G, i \in \mathbb{N}\}$. Using this set, define the $\alpha : G^* \to \wp(\Sigma_\alpha)$ function as $\alpha(\lambda) := \emptyset$ and $\alpha(\sigma a) := \alpha(\sigma) \cup \{a_{l(\sigma)+1}\}$, where $a \in G$, $\sigma \in G^*$ and $a_{l(\sigma)+1} \in \Sigma_\alpha$. For example, $\alpha(abcb) = \{a_1, b_2, c_3, b_4\}$. The indexing makes it possible to restore the original string uniquely, so this is a bijective method. Representing strings

✉ Gábor Román
romangabor@caesar.elte.hu

[1] Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary

with arbitrary length would require an infinite alphabet; however, with a fixed integer n, the restriction $\alpha(\sigma)|_{l(\sigma)\leq n}$ requires a finite subset of Σ_α of size only $n|G|$. The transition function for an L system is defined as $\delta : G^3 \to G^*$, so the system could produce a longer output than the input in a step of the computation. Taking this into consideration we could not use the previously established finite alphabet. We are going to overcome this by posing a restriction on the inferred L systems.

Myhill introduced the term linear-bounded automata, see [13], which is a deterministic Turing machine type working on a bounded tape segment (for more information, see [9]). We are going to show that these automata can be simulated by L systems, which work on filaments of fixed size. Linear-bounded automata have considerable computational power, see [10, 11], so the bounded version of L system which we create for their simulation has too. We will investigate the inference of such bounded L systems by constructing a polymorphic P system with target indication. The obtained polymorphic P system will be of size $O(n|G|^4)$, where G is the alphabet of the examined bounded L system and n is the maximal size of the filaments. The transitions of the bounded L system are processed in a teaching part by feeding the filaments of the transitions using an encoding into the polymorphic P system. After the teaching, the resulting polymorphic P system can reproduce the transitions of the examined sequence of filaments, when initiated with an encoded form of the initial filament.

2 Definitions

2.1 L systems

Let G be a finite nonempty set of symbols or cells, then G^* denotes the set of all strings or filaments, λ being the empty string. It is assumed that in the case of a cell at the end of the filament the outside environment will act on this cell just as if it was a cell in some state. A Lindenmayer system or L system is a tuple $L = (G, \delta)$ modifying filaments, where $\delta : G^3 \to G^*$ is the transition function, which indicates the new symbols by which the middle one is replaced for any three neighbouring symbols. This function can be extended to $\delta' : G \times G^* \times G \to G^*$, which indicates how a filament changes when it receives environmental input from both left (l) and right (r). The definition of δ' is, in this case, given as $\delta'(l, \lambda, r) := \lambda, \delta'(l, a, r) := \delta(l, a, r)$ and $\delta'(l, \sigma, r) := \delta'(l, \sigma' a_1, a_2)\delta(a_1, a_2, r)$, where $\sigma = \sigma' a_1 a_2$ with $\sigma' \in G^*$ and $a, a_1, a_2 \in G$. The computation of L is a finite sequence of filaments sequentially gained by the application of δ' starting from an initial $\omega \in G^*$ filament and ending in a halting configuration, where no symbol changes due to the application of δ'.

2.2 Polymorphic P systems

A polymorphic P system with target indication is the construct

$$\Pi = (\Omega, T, H, \mu, w_s, \langle w_{1L}, w_{1R}\rangle, \ldots, \langle w_{nL}, w_{nR}\rangle, h_i, h_o, \varphi),$$

where Ω is a finite alphabet of objects, T is the subalphabet of terminal objects, H is the finite set of membrane labels, μ is a tree structure consisting of $2n + 1$ membranes labelled with the elements of H, $w_s \in \Omega^*$ is the multiset giving the contents of the skin membrane, $\langle w_{iL}, w_{iR}\rangle$ are pairs of multisets giving the contents of membranes $iL \in H$ and $iR \in H$ $(1 \leq i \leq n)$, with $w_{iL}, w_{iR} \in \Omega^*$. We require that, for every $1 \leq i \leq n$, the membranes iL and iR have the same parent membrane. Furthermore, $h_i \in H$ and $h_o \in H \cup \{0\}$ are the labels of the input and the output membranes, respectively, where 0 denotes the environment. Finally, the $\varphi : \{1R, \ldots, nR\} \to \{in_h | h \in H\} \cup \{here, out\}$ mapping assigns a target indication to every right-hand-side membrane. The target indication is interpreted as in the case of conventional P systems. Note that with these restrictions, we effectively allow ignoring the hierarchical membrane structure for communication purposes.

The rules of Π are not statically given in its description but instead dynamically inferred for each configuration based on the contents of the pairs of membranes iL and iR. Thus, if in a configuration of the system these membranes contain the multiset u and v, respectively, then, in the next step, their parent membrane h will evolve as if it had the multiset rewriting rule $u \to v$ associated with it. If, however, in some configuration, iL is empty, we consider the rule defined by the pair $\langle w_{iL}, w_{iR}\rangle$ to be disabled, i.e. no rule will be inferred from the contents of iL and iR.

Polymorphic P systems evolve by applying the dynamically inferred rules in a maximally parallel way. A computation of a polymorphic P system Π is a finite sequence of configurations Π may successively visit, ending in a halting configuration in which no rules can be applied any more in any membranes. Like for other classes of P systems, the output of Π is the contents of the output membrane h_o projected onto the terminal alphabet T.

2.3 Linear-bounded automata

A linear-bounded automaton is specified by an $A = (Q, \Sigma, \Gamma, q_0, \#, F, \mu)$ tuple, where Q is the nonempty finite set of states, $\Sigma \subseteq \Gamma$ is the nonempty finite set of the input alphabet, Γ is the nonempty finite set of the tape symbols, $q_0 \in Q$ is the initial state, $\# \in \Gamma$ is a dedicated boundary symbol, and $F \subseteq Q$ is the set of final states; furthermore, the μ transition function is defined as

$$\mu : Q \times (\Gamma \setminus \{\#\}) \to Q \times (\Gamma \setminus \{\#\}) \times D,$$

where $D := \{-1, 0, +1\}$, and

$\mu : Q \times \{\#\} \to Q \times \{\#\} \times D$;

therefore, the # boundary symbol is not affected during computation. The automaton functions as follows. It is given as input a tape blocked into squares containing a string $\#x\#$, where $x \in \Sigma^*$. Before operation on the input, the state of the automaton is set to q_0. At the initial stage, it reads the left boundary symbol in the state q_0. In general, the computation of the automaton is the same as a deterministic Turing machine, except if it runs off the end of the given tape, designated by a boundary symbol, and at this time it finds itself in one of the final states of F, then by definition, the string x is accepted, or otherwise, rejected, by the automaton. The set of all strings accepted is the language accepted by the automaton.

3 Results

First, we show how linear-bounded automata can be simulated using L systems.

Lemma 1 *For every $A = (Q, \Sigma, \Gamma, q_0, \#, F, \mu)$ linear-bounded automaton with initial tape contents $\#x\#$, where $x \in \Sigma^*$, there exists an $L = (G, \delta)$ Lindenmayer system with $\delta : G^3 \to G$ and $\#_l^{q_0} x \#_r \in G^*$ initial filament, such that if A does not terminate, then L will not finish its computation, and if A terminates accepting (respectively, rejecting) with $\#y\#$ on its tape, where $y \in (\Gamma \setminus \{\#\})^*$, then L finishes its computation accepting (respectively, rejecting) with filament $\#_l y \#_r$.*

Proof Let $A = (Q, \Sigma, \Gamma, q_0, \#, F, \mu)$ be our original automaton and let $L = (G, \delta)$ be the constructed L system. Let $\Gamma_\# := (\Gamma \setminus \{\#\}) \cup \{\#_l, \#_r\}$ and $G := \{g, g^q, g^{q,d} \mid g \in \Gamma_\#, q \in Q, d \in D\}$ be the alphabet of L. We are going to mark the left and right boundary symbol with $\#_l$ and $\#_r$, respectively. Now $|G| \in O(|\Gamma||Q|)$, which is an alphabet of polynomial size.

If μ is defined as $\mu(p, a) := (q, b, d)$, then let $\delta(\cdot, a^p, \cdot) := b^{q,d}$, while if $\mu(p, \#) := (q, \#, d)$, then let $\delta(\cdot, \#_l^p, \cdot) := \#_l^{q,d}$ and $\delta(\cdot, \#_r^p, \cdot) := \#_r^{q,d}$ to rewrite the actual letter and change the actual state. Furthermore, let $\delta(\cdot, c, b^{q,-1}) := c^q$ when $b \neq \#_l$, and $\delta(b^{q,+1}, c, \cdot) := c^q$ when $b \neq \#_r$, to move the simulated head. Finally, let $\delta(\cdot, c^{q,d}, \cdot) := c$ when $d = \pm 1$ and $\delta(\cdot, c^{q,0}, \cdot) := c^q$ to remove the unnecessary information from the filament. The simulation of a step of automaton A can be seen in Fig. 1.

If the input of A is $x \in \Sigma^*$, then the initial filament of L is $\omega = \#_l^{q_0} x \#_r$, so the head is at the left boundary symbol and the state is q_0 at the start of the computation. When we arrive at a filament which contains either $\#_l^{q,-1}$ or $\#_r^{q,+1}$, then the computations are finished after the next application of δ'. If $q \in F$, then we finished the computation accepting otherwise rejecting. □

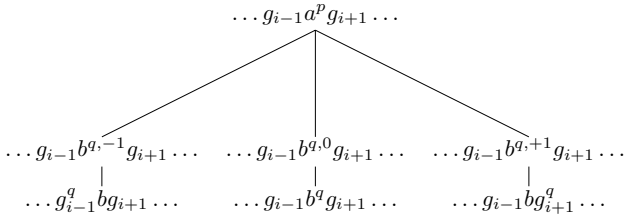

Fig. 1 Simulating a step of automaton A, where $g_{i-1}, g_{i+1} \in \Gamma_\#$. We rewrite a to b, change the state from p to q and step in one of the directions $\{-1, 0, +1\}$

Now we turn to the problem of inferring these bounded L systems with the aid of polymorphic P systems. The inference method will consist of two parts, first the teaching and then the recalling. In the teaching part, we apply a special β encoding to tell the polymorphic P system the transitions of the examined L system.

In the recall part, we input a filament of the given L system in α-encoded form, extended with indexed environmental inputs up to a fixed length.

The rules are stored as the transitions performed by a δ function. So in the membranes where the conversion happens, a rule is stored in the form of the three initial neighbouring symbols together with the resulting new symbol which replaces the middle one. Exceptional ones are the membranes responsible for the handling of the symbols at the ends of the filament. These membranes require only two neighbouring symbols to perform the replacement. So one membrane is responsible for the transformation of a symbol at one index only, but it requires the symbols at the neighbouring places. That is why distributor membranes multiply the symbol—which they are responsible for—into the proper membranes, where the rules are stored. The output of the transition arrives at the same membrane where the input came; therefore, the process will go on in a cyclic manner, see Fig. 2.

Let $l, r \notin G$ be the fixed left and right environmental input, respectively. Define $G_i := \{g_i | g \in G\}$, the elements of the L system's G alphabet indexed with a given $i \in \mathbb{N}$ integer. Let $\Sigma_0 := \{l_0\}$, $\Sigma_1 := G_1$, $\Sigma_i := G_i \cup \{r_i\}$ for $2 \leq i \leq n$ and finally $\Sigma_{n+1} := \{r_{n+1}\}$. We are going to work with multisets of fixed size in the recall part, so we introduce the alphabet extended with the indexed environmental symbols as

$$\Sigma_e := \bigcup_{i=0}^{n+1} \Sigma_i.$$

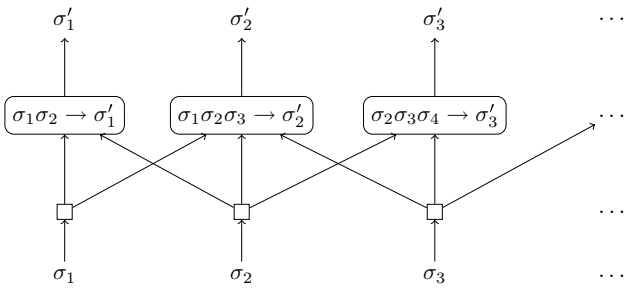

Fig. 2 Partial sketch of the steps performed during the recall part. A σ_i symbol is distributed into those membranes where its presence is required. After the stored rule is applied, it becomes σ'_i in the same membrane where it started out as σ_i, so the process repeats

Define $\alpha_e : G^* \to \wp(\Sigma_e)$ as

$$\alpha_e(\omega) := \{l_0, r_{l(\omega)+1}, \ldots, r_{n+1}\} \cup \alpha(\omega),$$

where $\omega \in G^*$ and $l(\omega) \leq n$. We extend the α encoded input with the indexed environmental symbols using this function.

The alphabet of the transitions is $\Sigma_\gamma := \{abc \to b' | a, b, b', c \in \Sigma_e\}$. We are going to encode the transitions with the $\beta : \wp(\Sigma_\alpha) \times \wp(\Sigma_\alpha) \to \wp(\Sigma_\gamma)$ function defined as

$$\beta(\{g_1\}, \{g'_1\}) := \{l_0 g_1 r_2 \to g'_1\},$$

where $g_1, g'_1 \in G_1$ and when $k > 1$, then

$$\begin{aligned}\beta(\{g_1, \ldots, g_k\}, \{g'_1, \ldots, g'_k\}) := &\{g_{i-1}g_ig_{i+1} \to g'_i | 1 < i < k\} \\ &\cup \{l_0 g_1 g_2 \to g'_1\} \\ &\cup \{g_{k-1}g_k r_{k+1} \to g'_k\},\end{aligned}$$

where $g_i, g'_i \in G_i$ for $1 \leq i \leq k$.

Proposition 1 *For fixed $n > 0$ integer, every $L = (G, \delta)$ deterministic Lindenmayer system with $\delta : G^3 \to G$, and $\omega_{ij} \in G^*$, $1 \leq l(\omega_{ij}) \leq n$ filaments with $1 \leq i \leq k$ and $1 \leq j \leq k_i$, where $\omega_{i1}, \ldots, \omega_{ik_i}$ is a correct computation of L for $1 \leq i \leq k$, there exists a polymorphic P system with target indication, which after processing the inputs $\beta(\omega_{ij}, \omega_{ij+1})$, where $1 \leq i \leq k$ and $1 \leq j < k_i$, the contents of the skin membrane initiated with $\alpha_e(\omega_{i1})$ changes as $\alpha_e(\omega_{i2}), \ldots, \alpha_e(\omega_{i(k_i-1)})$, finishing the computation with $\alpha_e(\omega_{ik_i})$ for $1 \leq i \leq k$.*

Proof Let

$$\Pi = (\Omega, T, H, \mu, w_s, \langle w_{1L}, w_{1R}\rangle, \ldots, \langle w_{mL}, w_{mR}\rangle, s, s, \varphi)$$

be the constructed polymorphic P system. Define the objects of the system as $\Omega := \Sigma_e \cup \Sigma_\gamma \cup \{t_1, t_2, t_3, t_4\}$, where $t_1, \ldots, t_4 \notin \Sigma_e \cup \Sigma_\gamma$ are symbols which will help us activate or inhibit rules. The output alphabet is defined as $T := \Sigma_e$.

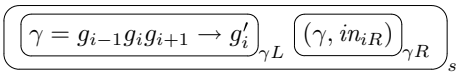

Fig. 3 Schematic for the $\gamma L, \gamma R$ rules in the skin membrane, with $\gamma \in \Sigma_\gamma$ having structure of $\gamma = g_{i-1}g_ig_{i+1} \to g'_i$, where $g_{i-1} \in \Sigma_{i-1}$, $g_i, g'_i \in \Sigma_i$ and $g_{i+1} \in \Sigma_{i+1}$ for $1 \leq i \leq n$

The labels, the membrane structure, the contents of the membranes and the target indication will be given as the proof goes. Where we do not state otherwise, the target of a rule is *here*.

The contents of the skin membrane are the iL, iR membranes, where $0 \leq i \leq n+1$, the gL, gR membranes, where $g \in \Sigma_e$, and the $\gamma L, \gamma R$ membranes, where $\gamma \in \Sigma_\gamma$. The iL, iR membranes are responsible for the processing of the ith cell in the filaments. The gL and gR membranes send the encoded symbols of the filaments into the iR membranes for processing; therefore, we are going to deal with these membranes in the discussion of the recall part. We store the rules and apply them in the iR membranes. Because of this, iL has no purpose, so $w_{iL} := \lambda$. The $\gamma L, \gamma R$ membranes send the encoded transitions into the appropriate iR processing membranes. The structure of these rules can be seen in Fig. 3.

An iR membrane for $1 \leq i \leq n$ contains the $i(g_{i-1}g_ig_{i+1})L, i(g_{i-1}g_ig_{i+1})R$ membranes, see Fig. 4. When a $g_{i-1}g_ig_{i+1} \to g'_i$ object arrives at iR, we store the left-hand side in $i(g_{i-1}g_ig_{i+1})L$ and the right-hand side in $i(g_{i-1}g_ig_{i+1})R$. In the beginning, every such membrane pair realises the $g_{i-1}g_ig_{i+1} \to g_i$ transition; therefore, the represented rule will not change the g_i object.

We set $\varphi(i(g_{i-1}g_ig_{i+1})R) := in_s$, so when the $g_{i-1}g_ig_{i+1}$ triplet arrives at iR, we transform g_i to g'_i and send it back to the skin membrane.

When a $\gamma \in \Sigma_\gamma$ object arrives in iR for $1 \leq i \leq n$, we store the transition using the membranes which can be seen in Fig. 5.

If we have not stored the given γ rule yet, then the $i\gamma\mathbb{L}L, i\gamma\mathbb{L}R$ rule sends the $t_1t_2t_3t_4$ objects into $i\gamma\mathbb{L}R$. This triggers the inclosed rules. The transition is rewritten in the appropriate membrane pair due to the presence of t_1 and t_2. The t_3 object is sent to the $i\gamma\mathbb{D}L$ membrane as a t_1 object for activation. There, the $i\gamma\mathbb{A}L, i\gamma\mathbb{A}R$ rule activates the $i\gamma\mathbb{D}L, i\gamma\mathbb{D}R$ rule, so when a γ object arrives again to the iR membrane, it will be simply erased. The whole learning mechanism is deactivated with the effect of the t_4 object, which is sent in the $i\gamma\mathbb{L}L$ membrane as t_1 for deactivation.

If we have already stored a given γ rule, then the $i\gamma\mathbb{L}L$ membrane will contain γt_1, so it will not be applied, whereas $i\gamma\mathbb{D}L$ will contain only γ, so the $i\gamma\mathbb{D}L, i\gamma\mathbb{D}R$ rule will delete the received γ object.

The $0R$ and $(n+1)R$ membranes are exceptions, there we only deal with sending the objects on the edges immediately

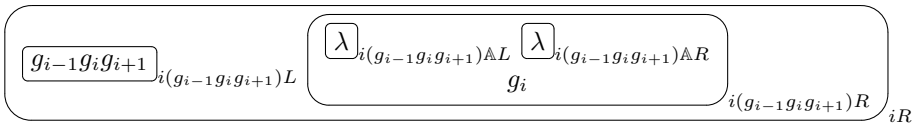

Fig. 4 Schematic for the $i(g_{i-1}g_ig_{i+1})L, i(g_{i-1}g_ig_{i+1})R$ transition storing membranes, where $g_{i-1} \in \Sigma_{i-1}$, $g_i, g_i' \in \Sigma_i$ and $g_{i+1} \in \Sigma_{i+1}$ for $1 \leq i \leq n$

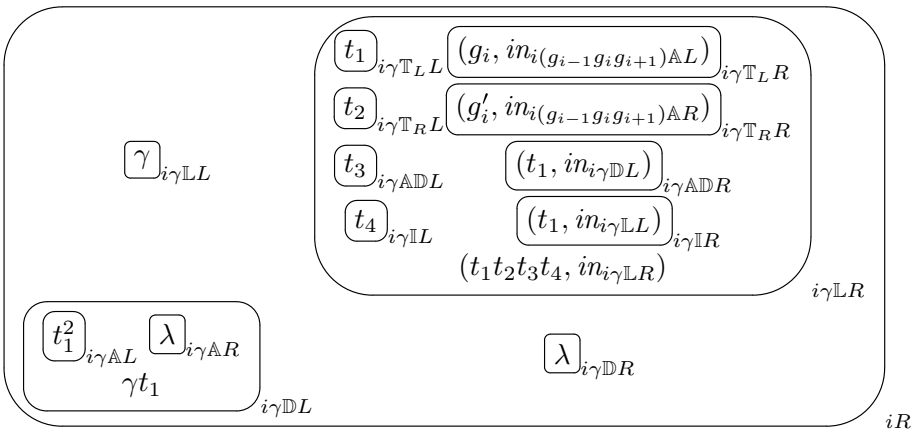

Fig. 5 Schematic for a iR membrane where $1 \leq i \leq n$ and the required membranes for the handling of a $\gamma = g_{i-1}g_ig_{i+1} \to g_i'$ transition, where $g_{i-1} \in \Sigma_{i-1}$, $g_i, g_i' \in \Sigma_i$ and $g_{i+1} \in \Sigma_{i+1}$

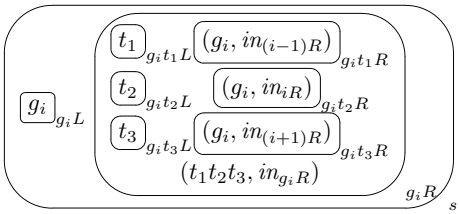

Fig. 6 Schematic for a g_iL, g_iR rule when $1 \leq i \leq n$

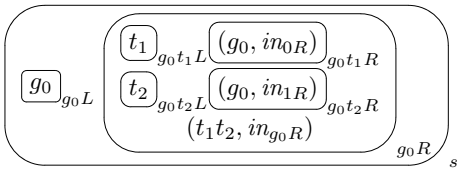

Fig. 7 Schematic for a g_0L, g_0R rule

back to the skin membrane. Because the $0R$ membrane receives g_0g_1, and we do not have to modify g_0, $0R$ contains only the $g_0 \to (g_0, in_s)$ and $g_1 \to \lambda$ rules for $g_0 \in \Sigma_0$ and $g_1 \in \Sigma_1$. Also, because the $(n+1)R$ membrane receives g_ng_{n+1}, and we do not have to modify g_{n+1}, $(n+1)R$ contains only the $g_{n+1} \to (g_{n+1}, in_s)$ and $g_n \to \lambda$ rules for $g_{n+1} \in \Sigma_{n+1}$ and $g_n \in \Sigma_n$.

During the recall part, we input the α_e encoded form of an $\omega_{i1} \in G^*$ initial string into the skin membrane, where $1 \leq i \leq k$. The input symbols are sent as triplets into the appropriate iR membranes for $1 \leq i \leq n$, using the g_iL, g_iR membranes, see Fig. 6. A $g_i \in \Sigma_i$ object triggers the g_iL, g_iR rule which sends $t_1t_2t_3$ into the g_iR membrane. Therefore, due to the presence of these objects, g_i is sent to the $(i-1)R$, iR and $(i+1)R$ membranes separately.

The symbols on the edges of the input are handled separately with the g_0L, g_0R and the $g_{n+1}L, g_{n+1}R$ membranes, see Figs. 7 and 8. Using the g_0L, g_0R rules, similar to the previous process, g_0 is sent into $0R$ and $1R$, while using the $g_{n+1}L, g_{n+1}R$ rules, g_{n+1} is sent to nR and $(n+1)R$.

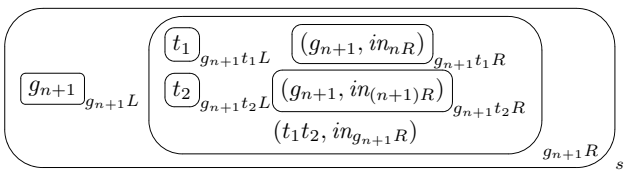

Fig. 8 Schematic for a $g_{n+1}L, g_{n+1}R$ rule

This way, an iR membrane receives the $g_{i-1}g_ig_{i+1}$ objects from the skin membrane for $1 \leq i \leq n$, while $0R$ receives g_0g_1 and $(n+1)R$ receives g_ng_{n+1}.

The rules in the iR membranes for $1 \leq i \leq n$ change the triplets according to the learnt transitions. Because the target of every $i(g_{i-1}g_ig_{i+1})R$ membrane is the skin membrane, the results of the $g_{i-1}g_ig_{i+1} \to g_i'$ transitions get back into the skin membrane. Even those objects get back into the skin membranes for which there was no transition presented during the learning phase, because in the beginning every $i(g_{i-1}g_ig_{i+1})L, i(g_{i-1}g_ig_{i+1})R$ rule realises the $g_{i-1}g_ig_{i+1} \to g_i$

rule. The g_0 and g_{n+1} objects get back to the skin membrane due to $0R$ and $(n+1)R$.

Finally, we count the size of the system. Because $|\Sigma_i| \sim |G|$, the number of $\gamma L, \gamma R$ membranes is $O(|G|^4)$ and because $|\Sigma_e| \sim n|G|$ the number of gL, gR membranes is $O(n|G|)$ in the skin membrane. In every iR membrane, there are $O(|G|^3)$ number of $i(g_{i-1}g_ig_{i+1})L, i(g_{i-1}g_ig_{i+1})R$ membranes, and $O(|G|^4)$ number of $i\gamma\mathbb{L}L, i\gamma\mathbb{L}R$ and $i\gamma\mathbb{D}L, i\gamma\mathbb{D}R$ membranes, so the number of membranes contributed by the iR membranes is $O(n|G|^4)$. Every membrane which we did not count contributes a constant multiplier to the membrane count, so we get that the total size membrane system is $O(n|G|^4)$ which is polynomial. □

4 Conclusions

We have showed that L systems which work on filaments of fixed size have considerable computational power because they can simulate linear-bounded automata. We have given a method to construct a polymorphic P system for a given G alphabet and n size, which polymorphic P system is of size $O(n|G|^4)$, and it solves the inference problem of L systems defined over the G alphabet working on filaments having size at most n.

It is still an open question that with what approach can one achieve the inference of arbitrary L systems using polymorphic P systems, so when we do not pose size limitations on the filaments of the examined L system. In this case, the indexing strategy surely will not work because it would introduce an infinite alphabet. It seems that one would have to resort to representing the filaments using the membrane structure of the polymorphic P system. This would require the introduction of membrane division, membrane dissolution, membrane creation and even dynamic targeting for polymorphic P systems (see the section of open questions in [2]).

It may be fruitful to approach the inference problem of Turing-complete systems with polymorphic P systems from a different angle, namely examining different types of machines. Queue automata make a very good candidate, see [9]. In this case, a great simplification can come from the fact that it can be easily decided which symbol(s) introduced which given change(s).

Acknowledgments Open access funding provided by Eötvös Loránd University (ELTE).

References

1. Alhazov A, Ivanov S, Rogozhin Y. Polymorphic P systems. In: Gheorge M, Hinze T, Păun G, Rosenberg G, Salomaa A, editors. Membrane computing, vol. 6501., Lecture Notes in Computer ScienceBerlin: Springer; 2011. p. 81–94.
2. Alhazov A, Freund R, Ivanov S. Polymorphic P systems: a survey. IMCS Bull. 2016;2:79–102.
3. Ben-Naoum F. A survey on L-system inference. INFOCOMP J Comput Sci. 2009;8(3):29–39.
4. Bulletin of the International Membrane Computing Society (IMCS). http://membranecomputing.net/IMCSBulletin/index.php
5. Feliciangeli H, Herman GT. Algorithms for producing grammars from sample derivations: a common problem of formal language theory and developmental biology. J. Comput. Syst. Sci. 1973;7:97–118.
6. Heinz J, Sempere JM, editors. Topics in grammatical inference. Berlin: Springer; 2016.
7. Herman GT. Computing ability of a developmental model for filamentous organisms. J Theor Biol. 1969;25:421–35.
8. Higuera C. Grammatical inference: learning automata and grammars. Cambridge: Cambridge University Press; 2010.
9. Kozen DC. Automata and computability. Berlin: Springer; 1997.
10. Kuroda SY. Classes of languages and linear-bounded automata. Inf Control. 1964;7:207–23.
11. Landweber PS. Three theorems on phrase structure grammars of type 1. Inf Control. 1963;6:131–6.
12. Lindenmayer A. Mathematical models for cellular interaction in development. J Theor Biol. 1968;18:280–315.
13. Myhill J. Linear bounded automata. WADD Tech. Note No. 60-165. Ohio: Wright–Patterson Air Force Base; 1960.
14. Păun G. Computing with membranes. TUCS Report 208 (1998). J Comput Syst Sci. 2000;61(1):108–43.
15. Păun G. Membrane computing. An introduction. Berlin: Springer; 2002.
16. Păun G, Rozenberg G, Salomaa A, editors. The Oxford handbook of membrane computing. Oxford: Oxford University Press; 2010.
17. Rozenberg G, Salomaa A. L Systems. Berlin: Springer; 1974.
18. Rozenberg G, Salomaa A. The mathematical theory of L systems. London: Academic Press; 1980.
19. Rozenberg G, Salomaa A. The book of L. Berlin: Springer; 1986.
20. Rozenberg G, Salomaa A, editors. Lindenmayer systems: impacts on theoretical computer science, computer graphics, and developmental biology. Berlin: Springer; 1992.
21. Sempere JM, García P. Grammatical inference: theoretical results and applications. In: 10th International colloquium, ICGI 2010, Valencia; 2010.
22. The P systems website. http://ppage.psystems.eu/

Gábor Román is a Ph.D. student of Eötvös Loránd University, Budapest, at the Faculty of Informatics, since 2015. His main research areas are computational number theory and analytic number theory, but he is also interested in non-conventional computing.